A versatile timing unit for traffic shaping, policing and charging in packet-switched networks

L. Sarakis a,b,*, N. Moshopoulos c, D. Loukatos a, K. Marinis a, P. Stathopoulos a, N. Mitrou a

a National Technical University of Athens, School of Electrical and Computer Engineering, Heroon Polytechniou 9, 157 73 Athens, Greece
b National Centre for Scientific Research “Demokritos”, Institute of Informatics and Telecommunications, 15310 Aghia Paraskevi, P.O. Box 60228, Athens, Greece
c ATMEL HELLAS S.A., IC Group, Papazahariou 48, Ag. Dimitrios, Athens, Greece

Received 24 May 2006; received in revised form 31 January 2007; accepted 16 August 2007
Available online 14 September 2007

Abstract

Timing has a key role in several traffic control functions encountered in modern packet-switched networks. In order to be effective, a timing unit must provide fine resolution, be simple to implement and scale well with the number of controlled traffic streams. This paper addresses the design, implementation and evaluation of a timing unit that can support accurate and efficient implementations of traffic shaping, policing and charging in packet-switched networks. The timing unit is implemented in hardware and, therefore, overcomes constraints associated with software-based timers. It accommodates a pool of independently-clocked timers and counters, organised in timing blocks, and, consequently, is able to support, in parallel, traffic streams with diverse timing requirements. The design supports shaping and policing through token buckets, leaky buckets and a scheme, variation of the token bucket, that aims at providing statistical quality of service guarantees by exploiting the effective rate concept. Charging is supported by dedicated counters that measure the utilisation of the effective rate. The granularity of the timing unit is adjustable in run-time to adapt to changes in the rate parameters of the shaping and policing functions. The validation of the timing unit is done through the development of a prototype board consisting of programmable hardware and embedded software blocks. The temporal resolution of the timing unit and the advantages of the hardware/software co-design are experimentally evaluated.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Temporal resolution; Hardware/software co-design; Token bucket; Effective rate; QoS

1. Introduction

Packet processing inside networking elements is at the moment relatively limited, so as not to impair performance. Over time, as computing power is becoming cheaper, functionality that will permit the control of the hardware and the low-level networking software will be deployed. This enhanced functionality, which targets at delivering better services, control and management capabilities to both end-users and network service providers, and the way it is provided, is the goal of the ongoing research in the field of network programmability [1]. The key features of programmable networks are: (a) modular architecture of hardware and protocols and (b) open interfaces that facilitate the programming and customized configuration of hardware and software modules.

In the lowest level, network programmability means ability to dynamically configure the hardware and the
associated control functions like routing, packet classification, scheduling, admission control, bandwidth allocation, traffic policing and shaping. These functions are essential for providing service differentiation and enhancing the IP best-effort service model with Quality of Service (QoS) guarantees and, thus, they have been addressed in both the Integrated Services (IntServ) [2] and the Differentiated Services (DiffServ) [3] frameworks introduced by the IETF (Internet Engineering Task Force). Network nodes inside IntServ domains implement functionality for traffic control, admission control and resources reservation on a per flow basis. The tremendous overhead that may be placed on core routers led to concerns about the scalability capabilities of the framework. In DiffServ, on the other hand, the basic architecture assumes that complex traffic classification and conditioning functions (metering, marking, shaping, policing) are located only in the network’s edges (however, deployment of these functions in the interior of the network is not precluded). By transferring the complexity from core elements to boundary nodes, the framework has the potential to scale well with the increasing volume of IP traffic.

In the heart of many traffic control functions there is a complex of clocks, timers and counters supplying for each controlled stream a time base and the necessary counting facilities. Throughout the paper such a complex will be referred to as timing unit. For a large number of controlled streams, the implementation of this timing unit may be critical for the efficient operation of the network element that hosts it. The number of controlled streams may range from a few thousands in core routers of IntServ domains to a few hundreds or tens in egress routers of corporate intranets (source networks) and boundary nodes of DiffServ domains, and to about a dozen or less inside DiffServ domains (where the streams are usually classified according to the DiffServ codepoint).

In commercial networking products, traffic management and control functionality is implemented in hardware (common for ATM adapters), software or a combination of them (for example, see [4]). Furthermore, sophisticated network processing engines (network processors), which integrate packet processing, classification and traffic management on a single chip, have gained much attention since they have the potential to deliver the computational power that is needed for packet processing in gigabit rates. However, the complicated designs that are usually found in commercial products today, targeting either the ATM or the IP domain, basically correspond to proprietary hardware/software solutions and, thus, are of relatively limited value in experimental laboratory-based platforms, where full control over hardware and software is required.

An extreme example of a programmable network component (allowing full control over hardware and software) is a general-purpose computer, equipped with the appropriate network cards and a slim operating system (e.g., Linux), acting as a network node in a QoS-aware installation. Such configurations, involving standard end systems and open-source software code, are typical in networking research and are widely deployed in experimental platforms. In these setups, traffic control operations are implemented on-board and the timing functionality is provided by software-based timers. In such cases, implementing the counters/timers in software may be quite an overhead for the central processing unit (CPU), thus an off-board implementation can be a judicious alternative. The motivation behind our work was the intention to overcome computational overheads and timing inaccuracies originating from software implementation of timers and come up with a design that has the potential to address distributed traffic control at the network interface adapters.

In this context, the paper presents the design, implementation and evaluation of a timing unit that can accommodate the stringent timing requirements of traffic shaping, policing and charging in packet-switched networks. The timing unit, which is implemented in hardware, is structured into a set of independently-clocked timing blocks. Each one of these blocks, consisting of counters and timers sharing a common clock, is associated with a different controlled stream. The design satisfies the requirements for a modular architecture of timers and counters allowing their dynamic clock configuration. The timing blocks have been properly dimensioned (in number of counters and timers) to support a traffic shaping scheme that is able to enforce specified effective rates [5,6] to the shaped streams [7]. This data spacer is more complicated than the well-known token bucket (TB) and leaky bucket (LB) schemes [8], and, thus, introduces extra system challenges. In Section 2 it will become apparent that this spacer can be regarded as a “modified” token bucket (modTB) scheme with peak rate control and that any design that can address this shaper can also be used for the traditional TB and LB. The complex of counters and timers used for shaping can be equally used for policing according to the LB, TB and modTB. Charging, on the other hand, introduces extra counters, the role of which depends on the factors that are used to calculate charges. In the proposed design of the timing unit, charging is based on effective rates and takes into account the call duration and the utilization of the effective rate.

The validation of the timing unit and the demonstration of an off-board implementation of traffic shaping, policing and charging are done through the development of a PC board. The traffic control operations are analyzed into a sequence of tasks, which are then mapped onto embedded software and hardware. The timing unit and the functionality that is better accommodated by hardware are incorporated into Field Programmable Gate Array (FPGA) logic, while the remaining functionality is undertaken by a microcontroller. The prototype board is used in experiments that evaluate the temporal resolution of the timing unit and demonstrate the advantages of supporting complicated traffic shaping schemes through an off-board implementation.

The remainder of the paper is structured as follows. Section 2 addresses the role of timing in traffic shaping, polic-
scheduling and charging and presents their implementation requirements. Section 3 discusses the impact of software-based timers on the precision of data spacing and provides an overview of software-based techniques that have been proposed to increase the resolution of the system timer. Section 4 presents the architecture of the programmable timing unit and addresses scalability and design considerations. Section 5 introduces the architecture of the peripheral board, describes the functionality of the hardware and software blocks and comments on the system overheads. The resolution of the timing unit and its use in traffic shaping are experimentally evaluated in Section 6. Finally, Section 7 concludes the paper and presents directions for further research.

2. The role of timing in traffic shaping, policing and charging

In this section, we first discuss traffic shaping and present its implementation requirements in terms of timers and counters, and then comment on the role of timing in traffic policing and charging. Traffic shaping is analysed in greater depth because its implementation is more complicated than that of policing or charging. The token bucket and leaky bucket are discussed first to let the reader track the complications of the modTB more easily.

2.1. Traffic shaping with token bucket and leaky bucket

Traffic shaping in packet-switched networks are usually accommodated by LB, TB or combinations of them (see, for example [9,10] and references therein). The LB scheme, originally proposed in [8] for cell-switched architectures, targets mainly at policing an incoming traffic; however, with slight modifications it can be used to shape a bursty stream into a regular flow of cells. LB can be also used to police or regulate datagram traffic; in this case, the rate and the size are measured in bytes. The TB, mainly proposed for regulating or policing IP traffic, is a variation of the LB in the sense that although it allows bursty traffic to pass through it bounds the length of the burst.

A generic TB (GTB) scheme with LB peak rate control is shown in Fig. 1. The term “generic” is used to describe algorithms that do not necessarily involve the linear data spacing law associated with the traditional TB; the terminology will be fully justified after the discussion of modTB. This configuration provides for a modular (in the sense that it consists of subunits that can be activated and deactivated on desire) and flexible shaping scheme that is able to control not only the length of the burst but also the maximum rate at which this burst (sequence of packets) is transmitted.

Throughout the discussion it is assumed that the size of the packets (data units) and the time are both quantized into integer values expressed in data atoms (DA) and time slots (TS), respectively. The time slots and, consequently, the data atoms transmitted in them can be freely chosen to correspond to bits, bytes or any multiple of them (even ATM cells).

An incoming packet of size \( v \) joins the GTB’s input buffer and awaits permission to move on. The buffer is properly dimensioned so as to comply with the maximum latency a packet is allowed to experience inside the shaper. The time a packet spends in the input queue is determined by the packet size, the GTB algorithm (GTBA), the token arrival rate, \( f \), and the number of accumulated tokens (one token is sufficient for one DA). After acquiring the necessary credit, the packet moves to the LB queue. The time the packet spends in this queue is determined by the peak rate enforcement (PRE) unit and is at most \( v/p \) TS. After that, the packet is managed by the scheduler and sent to the network. The LB rate \( p \) is usually greater than \( f \) and thus, no packet is discarded as soon as both units have buckets of the same size, say \( b \).

The TB can be regarded as a special case of the general module of Fig. 1. In this case, there is no PRE unit and the GTBA involves a linear data spacing law in the sense that, excluding burst transmissions, the enforced silence is a linear function of the packet size.

2.2. Traffic shaping with modified token bucket

Alike TB, this traffic shaping scheme [7] can be also viewed as a special case of the general module depicted in Fig. 1. However, unlike TB, it takes advantage of the PRE unit and uses a different GTBA. The modTB algorithm is briefly described below and contrasted with the TB algorithm. We assume for the moment that there are always packets waiting at the shaper’s queue, i.e., the shaper works under a maximum utilization scenario.

In case of TB, when a packet arrives at the shaper’s queue, it is delayed for a time \( s \) that is proportional to its size \( v \) and inversely proportional to the bucket replenish-
moment rate $f$, i.e., $s = v/f$. In the modTB algorithm, on the other hand, the packet is held in the queue for time $s$ given by

$$s = \left( \frac{\left( \frac{e^z - 1}{z} \right) \cdot \frac{1}{f}} \right), \quad z = |qv|$$

(1)

where $f$ and $q$ are parameters of the shaper. It has been proven [7] that this shaping scheme can provide statistical QoS guarantees. This is the case when $N$ independent traffic streams (corresponding to either individual or aggregate flows) enter a statistical multiplexer having a service rate $C$ and working on first-come-first-served basis. The input streams are shaped according to (1) prior to multiplexing at the output queue. Each one of the $N$ shapers can have its own rate parameter ($f_i$, for the $i$th shaper), while all of them share the same parameter $q$. If the multiplexer’s service rate fulfills $C \leq \sum_{i=0}^{N-1} f_i$, the distribution of the multiplexer’s queue-length is upper bounded according to the formula: $\gamma e^{qv} \leq \Pr\{\text{queue\_length} > x\} \leq e^{qv}$, $\gamma \in (0, 1)$. The parameter $f$ corresponds to the effective rate that is assigned to each stream, while the parameter $q$ determines the target slope (in log-arithmetic scale) of the queue-length complementary probability distribution function, being essentially the QoS parameter of the system.

In the non-maximum utilization scenario there exist time periods when the shaper’s queue is idle. During these periods, tokens are accumulated with fixed rate (same for all shapers) equal to one token per time slot. Each shaper is associated with a credit (token) bucket with parameters $c\_\text{max}$ (depth) and $c\_\text{init}$ (initial content). Upon packet arrival, the amount of credit that is removed is given by (1). If the new value of credit is less than zero, a silence is enforced until the credit counter reaches $c\_\text{init}$. Otherwise, the enforced silence is $v/f$. In any case, the minimum enforced silence is $v/f$ keeping the normalized load through the multiplexer less than unity, in cases of large credit thresholds.

2.3. Timing-specific requirements of shaping

The timers and counters associated with the general shaping unit of Fig. 1 are depicted in Fig. 2. The GTB is associated with a rate and a bucket and, consequently, its implementation requires one clock, of frequency $fT^{-1}$, and one counter, credit, saturating at $b$. The counter holds the accumulated tokens, while its top value corresponds to the maximum allowed burst. The value of credit is reduced upon arrival of a new packet. Similarly, the LB requires a clock, of frequency $pTS^{-1}$, and a counter, $pcredit$, saturating at $v_{\text{max}}$. The top value of the counter corresponds to the maximum size of incoming packets. The value of $pcredit$ is set to zero after a packet has been released by the shaper and left to scheduler. In this way its value at any instance (unless it is saturated) reflects the time elapsed since the transmission of the previous packet and, consequently, the time the incoming packet must be delayed so as to adhere to the maximum transmission rate.

The size, $v$, of the incoming packet is passed as a parameter to both GTBA and PRE subunits. In this way the two different data spacing functions can be carried out in parallel. The outcome of GTBA and PRE are the values of the calculated delays $s$ and $ps$, respectively. The maximum of $s$ and $ps$ is the total delay the packet should encounter inside the shaper. This conformance time (normalized to an appropriate time base) is applied by a silence-enforcement timer, $Ts$, fed with a clock of frequency $f_{Tx}$. Upon exit from the shaping unit, the packet is handled by the scheduler, the role of which is to resolve conflicts arising when multiple packets, from different streams, become eligible for transmission. Therefore, it can be either a simple first-come-first-served scheduling or a more complicated fair arbitration scheme (for example [11–15]).

Having described the timing-specific requirements of the general shaping unit, we now place our focus on the modTB. The description in Section 2.2 showed that the modTB constitutes a non-linear data spacer (the non-linearity is highlighted in (1)) with peak rate control. Thus, its implementation involves both parts of the general shaping unit (Fig. 2). The modTB algorithm, as described above, uses a GTB with filling rate 1 token per TS and thus, its content corresponds to the accumulated time tokens. Accordingly, (1) gives the calculated silence $s$ and involves a division with the parameter $f$. This division can be eliminated by changing the bucket replenishment rate to $f$, so that now the bucket content (and accordingly $c\_\text{init}$ and $c\_\text{max}$) corresponds to data tokens. The implementation of this part requires a clock of frequency $fT^{-1}$ feeding a credit up-counter (Fig. 2), similarly to the TB. However, the differ-

![Fig. 2. The general shaping unit and the underlying counters and timers.](image-url)
ence from TB is that the modTB algorithm calculates $s$ according to size $v' = (e^2 - 1)v/z$, instead of $v$. The PRE unit enforces the minimum silence $ps = v/f$ and also requires a clock of frequency $fTS^{-1}$ (i.e., $p = f$ with respect to Fig. 2). Since both credit and pcredit counters use the same clock, the pcredit counter can be eliminated and its functionality can be easily supported by credit (in implementation terms this means that the modTB algorithm must introduce and manage an additional threshold). Summing up, the implementation of the modTB requires one counter, credit, with clock $fTS^{-1}$ and one timer, $Tx$, with clock $fTx$.

A comment is due at this point regarding Fig. 2. The two counters and the single timer have been associated with three different clocks mainly for two reasons. First, this arrangement provides the designer with the freedom to control the accuracy of each part independently of the others, according to the implementation requirements. Second, a real system hosting this shaping module may impose different constraints on counting and timing functions.

To illustrate this, consider the case where shaping is accommodated in software running on top of the operating system of a networking node. We further keep the discussion focused on general-purpose computers running the Linux operating system, since this combination is a natural choice for nodes participating in experimental networking platforms. In such cases, the values of credit and pcredit, and correspondingly the silences $s$ and $ps$, can be computed with high precision either by using the time provided by the system or by reading the value of the timestamp counter found in later Pentium processors (similar counters exist in other processor architectures). The system time has near microsecond resolution in Linux. The resolution of the 64-bit timestamp counter is even higher and depends on the clock frequency of the chip. On the other hand, the resolution of the silence-enforcement unit is much coarser since it accommodates a scheduling task that inherently depends on the resolution of the operating system timer. This resolution has a default value of 10 ms in Linux (i.e., $f_{Tx} = 100$ Hz) to balance between response time and system overhead mainly caused by context switching.

The general shaping module presented in Fig. 2 follows the conventional design approach, according to which the shaping function is separated from the link scheduling function [16]. This separation highlights the timing requirements of the shaping function and facilitates the mapping of these requirements onto the timing unit that will be presented in Section 4. Furthermore, since the timing unit is to address not only traffic shaping but also policing and charging, such separation is well motivated. For a design approach that integrates traffic shaping and link scheduling see [9].

2.4 Traffic policing and charging

Traffic policing is, in practise, accommodated by slightly modified versions of the corresponding shapers and further-more, it does not require facilities for task scheduling. Thus, the analysis of traffic shaping, presented in the previous sections, can be used, in general, for policing as well.

On the contrary, charging has looser (if any) requirements in terms of time keeping. For example, the flat rate model [17] that currently dominates Internet pricing does not require any timing facility, while charging based on call duration can be accommodated in a relatively simple way without the use of sophisticated timing modules. However, since charging has been identified not only as a means for infrastructure cost recovery, but also as a key factor towards better traffic management and control, efficient usage-based pricing schemes are brought into play (see [18] for a review of these schemes). Towards this direction, several metrics, which can be used to calculate charges, have been introduced and thoroughly investigated (e.g., utilization, resource allocation, call duration, access bandwidth, number of hops between source and destination, etc).

Since the effective rate constitutes a measure of resource requirement, it can naturally be used for charging (the idea was introduced in [19]; see also [20]). In this paper, we do not intend to argue on the suitability of the effective rate concept for charging; we rather investigate how a timing unit, used for policing and shaping according to an allocated/enforced effective rate, can be enhanced with facilities for more complete accounting and charging. As far as such a timing unit is concerned, these facilities may include counters capturing details that are relevant to bandwidth utilization and, therefore, to charging. Additionally, in order to be of a greater value, a versatile timing unit must be able to track those details that can not (or is very hard to) be captured by the network’s accounting and charging entities. Recalling the implementation of the modTB, it is reasonable to include counters that accommodate the time interval during which credit is saturated or $Tx$ is idle. Both of these metrics evaluate the degree of utilization of the shaping/policing unit; consequently, they can be used to add fairness to a charging scheme that works on the basis of the allocated effective rate and call duration.

3. Impact of software-based timers on the precision of data spacing

The implementation of traffic control functionality in software depends strongly on the temporal resolution of the operating system of the network node. This resolution has impact on the time difference between the desired (scheduled) and the actual execution of an event. In cases where this difference must be kept small, a temporal resolution of 10 ms, common in general-purpose operating systems (like the standard Linux), is not acceptable. This resolution provides a good compromise between responsiveness and context switching overheads; however, it is inadequate for applications that have stringent timing demands.

At the level of packet transmission scheduling, the operation of data spacing devices resembles that of traffic
generators. Both operations address the enforcement of a silence between two consecutive transmissions. The difference is that in traffic generation, the silence (i.e., inter-packet distance) is calculated according to the desirable traffic profile, while in traffic shaping the silence is derived by the shaping algorithm. After the calculation of silence, these two functions are almost indistinguishable. Therefore, the evaluation of the impact of system timers on the realization of traffic shaping can be done through the performance assessment of traffic generators that rely on software-based timers.

For this purpose, two traffic generators running on top of the Linux operating system are used. The first of them (the tg [21] traffic generator) is an example of a relatively simple implementation that relies on the existing timing structures of the operating system. On the contrary, the second tool (the rude [22] traffic generator) incorporates complicated scheduling techniques and strives to achieve finer resolution. The two traffic generators were set to transmit fixed-length packets (of size 200 bytes) with inter-packet distance of 200 μs.

The inter-packet distances were measured using the tcpdump network traffic analyzer [23] and the probability distribution function (PDF) of the difference between actual and desired distances is depicted in Fig. 3. Visual inspection of the results shows a clear distance between the ideal and the experimental curves for both tools. As far as the tg tool is concerned, 87% of the inter-packet distance samples lie in the interval 275 ± 5 μs and 3% of them in the interval 110 ± 5 μs. The rest of the samples are scattered in intervals around 305 μs and 530 μs. The performance of the rude tool is slightly better, since 75% of the samples lie in the interval 225 ± 5 μs and 20% of them in the interval 75 ± 5 μs (the rest of the samples are around 250 μs and 470 μs).

The results show that software-based timers represent a limitation in the realization of precise scheduling that has severe impact on the profile of the transmitted traffic. Moreover, in case there are many active timers, the handling of the timers itself (set, handle and cancel timer operations) constitutes a significant source of system overhead (for a thorough investigation of the overheads inserted in kernel level, see [24]). Thus, for efficient support of a large number of concurrently active streams there is a need for scalable timer support. Towards this end, much research effort has been put on refining the system granularity and delivering an accurate timing unit that can be used in a wide range of real-time applications. In the following, we skip proprietary real-time operating systems and discuss briefly three approaches to providing Linux with a better than default temporal granularity.

The simplest way to increase the temporal resolution of Linux is to program the timer chip to produce interrupts still in a periodic fashion, but at a rate larger than the default of 100 Hz. Although this would decrease the time difference between scheduled and actual execution of an event, it may introduce unnecessary overhead to the CPU and result in lengthier execution cycles. A more sophisticated approach that bounds the delay between the scheduling of a task and its execution (usually in the range of a few microseconds) is that of Real-Time Linux (RTLinux) [25]. RTLinux treats the Linux operating system kernel as a low-priority task executing under a small real-time operating system. Although this approach is valuable for applications that specifically fit the RTLinux model (i.e., applications that are split into real-time and non-real-time parts) it may not be suitable for applications that access the networking subsystem of Linux (like the traffic control functions) because this task is not supported by RTLinux [26].

An interesting approach that uses the clock device in the one-shot mode has been proposed in [26]. In this approach, the clock device instead of generating interrupts in a periodic fashion is programmed to interrupt the CPU in time to process the earliest scheduled event. In this way every time-critical event is indicated to the CPU through an interrupt. However, the price paid is the overhead associated with the reprogramming of the clock in each interrupt, since this task can be relatively slow for x86 generic motherboards [27]. In order to keep kernel subsystems working properly, there is provision for generating additional timer interrupts at regular clock intervals (e.g., 10 ms) just like in standard Linux. Thus, in any time interval the total number of interrupts issued to CPU equals the sum of the interrupts that would have been generated by the standard Linux and the interrupts that are generated by the time-critical events. It will become apparent in the following section that there is a close resemblance between this mechanism and the express response structure of our timing unit, as far as the notification of events is concerned. In both cases the events are communicated through interrupts and, furthermore, the total number of generated interrupts is the same. The difference is that our approach uses a separate interrupt line to notify the CPU and does not interfere with the system timer.

4. The programmable timing unit

In this section we present the architecture of a programmable timing unit that can accommodate traffic shaping,
policing and charging and elaborate on scalability and design issues. We also present the interaction between this and an algorithm-execution and control (AEC) unit, the role of which is to control the timing unit and execute the algorithmic parts of the traffic control and accounting functions of interest.

### 4.1. Architecture and interface with the AEC unit

The design of the programmable timing unit aims at achieving several key objectives: (1) support for traffic shaping, policing and charging through programmable counters, (2) adherence to a modular architecture capable of supporting, in parallel, several traffic streams through individually clocked timing blocks, (3) enforcement of silence without relying on external (i.e., host-based) clocking mechanisms and (4) fast communication of time-critical events to host.

Fig. 4 shows the architecture of the programmable timing unit, as well as the interfaces with the AEC module and the external CPU. The AEC block can be implemented in hardware as well as in software. In the second case, the code-execution platform can be either the CPU of the network node or a dedicated processor.

The programmable timing unit consists of a master clock, a clock distribution logic, a number of timing blocks (each of them consisting of a set of counters, timers and registers), a block for event notification (express response structure – ERS) and a timestamp counter. The clock distribution logic delivers $N$ clock signals and, consequently, is able to support, in parallel, $N$ separate traffic streams. The parameter $N$ is usually bounded by implementation constraints. Every controlled stream is associated with a timing block that comprises three registers, three counters and one timer. Each of the registers $clock\_dividers_i$, $0 \leq i \leq N - 1$, is user-configurable in run-time and its value represents the divider (prescaler) of the master clock that yields the desired clock, $clock_i$, for the counters and the timer of stream $i$.

The architecture of the programmable timing unit has been mainly motivated by the requirement to accommodate shaping/policing through modTB (and, inherently, through TB and LB), as well as charging through dedicated counters. For the TB and modTB, the counter $credit_i$ holds the accumulated credit necessary for transmission (instantaneous content of the bucket) and in the general case is loaded with an initial value $c\_init_i$. Register $c\_max_i$ corresponds to the depth of the bucket. For LB rate control the operation of $credit_i$ is slightly different. In this case, after the transmission of a packet to network, $credit_i$ is reset and then left to run free up to $c\_max_i$ that now corresponds to the maximum packet subjected to traffic control. In shaping mode, the timer $T_{xi}$ enforces the silence required before (or after) transmission. In policing mode, this timer is used to notify the host of the outcome of the policing function in a timely manner. The $time\_stamp$ counter is fed by the master clock and serves as a local time reference.

Charging is accommodated by the up-counters $duration_i$ and $charging_i$, both of which are associated with $clock_i$. By this clock assignment it is ensured that a stream will be charged according to its share of the available bandwidth. The first counter corresponds to time-sensitive charging, i.e., charges are based on connection time (the counter is running as far as the timing block $i$ has been allocated clock and until it is reset). The second counter corresponds to usage-sensitive charging and is running as far as the credit counter is not saturated (then stops). In this way, when a stream is idle it should not be charged, since the bandwidth it releases can be delivered to active streams (e.g., streams of best-effort traffic).

The ERS is used to collect the time-critical responses and communicate them to host in an efficient and timely manner. It does so by providing centralized monitoring of the timers’ status into a single event register. ERS is used mainly by traffic shaping to signal the end of the enforced
silence, but can be also used by policing to communicate
the outcome of this function.

As it may be expected, the design of this block raises the
following question: interrupts or polling? The use of inter-
rupts results in improved response times and, consequently,
in improved performance of traffic control operations, pro-
vided that the interrupt rate can be handled by the host.
However, improved performance of traffic control opera-
tions may come at the cost of deteriorating the perform-
ance of other tasks running on the network node. Polling, on
the other hand, preserves the code-execution
flow in the host CPU and results in simpler manipulation
of the responses and in lower CPU load in cases of large
responses rate. The _event_threshold_ register, depicted in
Fig. 4, is part of the ERS and its role will be described later.

The AEC unit is independent of the underlying timing
unit and is responsible for initializing and managing the
timing unit, as well as executing the control and accounting
algorithms. The execution unit interfaces with both the
timing unit and the CPU that triggers the execution of
the traffic control functionality. At initialization stage,
the AEC block loads the _clock_divider_, _c_init_, and _c_max_
registers with appropriate values. At run-time it reads
and updates _credit_, sets _Txi_, reads _duration_, and _charging_,
and possibly updates the three last registers. The remaining
_time_stamp_ counter can be directly accessed by the external
processor.

4.2. Scalability and design considerations

The timing unit presented in Fig. 4 accommodates traffic
control and accounting functions by assigning a separate
timing block to each of the controlled input streams. Each
of these timing blocks is fed by a separate clock delivered
by the clock distribution logic. An efficient implementation
can be realized if the frequency of this clock is programm-
able and associated with the rate parameter of the shaping/
policing schemes. However, the need for individually pro-
grammable clocks may leave room for concerns about
the effectiveness of the design. One of these concerns has
to do with the ability of individual clocks to adhere to speci-
fied rate parameters and is raised by the fact that the cur-
rent design supports only integer clock dividers. However,
there are two ways to overcome this problem and provide
close matches of the desired clocks.

The first is to use fractional dividers [28] at the cost of
increased design complexity. (The implementation of a
fractional divider using purely digital resources requires a
much higher reference frequency in order to achieve the
granularity needed for a fractional frequency. Such timing
capabilities are device-dependent.) The second is to choose
suitable values for the DA quanta of the controlled streams
so as to approximate the clocks by using integer dividers.
In the following, we turn our focus on issues that are
related to the scalability of the design, i.e., the capability
of supporting, in parallel, a varying (and possibly large)
number of individual traffic streams.

The timing unit consists of three major functional
blocks: the clock distribution logic, the ERS block and
the pool of timing blocks. We elaborate on each of these
blocks in turn. The scalability concern with clock distribu-
tion logic has to do with the number of different clocks it
can provide. In hardware terms, this objective is mainly
related to the ability of the design to meet timing specifica-
tions as well as specific constraints imposed by the adopted
implementation technology. For example, a common con-
straint in FPGA designs, unlike Application Specific Inte-
grated Circuit – ASIC, is that every clock source must be
associated with a clock buffer. Given that today’s FPGA
chips host only a small number of such buffers, this, inev-
itably, limits the number of clock signals that can coexist.
Although the actual number of timing blocks that can be
supported strongly depends on the target hardware plat-
form, meeting the design goal of the clock distribution
logic is, in general, getting harder as the number of these
blocks increases.

However, taking into consideration the fact that a mod-
erate number of such blocks (e.g., a few tens) are feasible,
an efficient workaround can be realized by grouping
together traffic streams with similar rate requirements
and assigning to each one of these groups a _different_ clock.
The frequency of each clock will be determined by the
requirements (time granularity) of the stream with the larg-
est rate within a group. The assignment of clocks to differ-
ent groups (and, accordingly, the classification of flows into
groups) can be done either in a static/predefined manner or
in a dynamic way during run-time, based on the capabili-
ties of the clocks-management software. Intelligent clock
assignment, which evenly distributes flows among different
groups, has the potential to mitigate performance deterio-
ration caused by flows competing for common resources
(timing blocks). This enhancement, which can be easily
supported by the programmable timing unit, has the poten-
tial to alleviate implementation overheads associated with
the support of traffic streams with _very different_ rates (these
overheads have also been addressed in previous works; see
for example [9]). The use of groups eliminates the need for
having a separate clock for each stream but adds complex-
ity to the AEC unit due to the management of the individ-
ual streams within a group and the calculation of their
conformance times.

The timing blocks consist of counters, registers and tim-
ers. Thus, the scalability issues concerning these blocks are
mainly associated with silicon area consumption. Clearly,
the hardware implementation technology (FPGA, ASIC,
or other), the design software tools and the capacity of
the chip have impact on the number of supported streams.
For specific target hardware, the support of additional tim-
ing blocks can be realized by reducing the amount of
shared hardware resources (number of flip-flops and mem-
ory cells) that are consumed by each timing block. This
requires careful dimensioning of counters, registers and
timers and possibly a redesign of the timing blocks that
moves parts of the functionality to software. In this con-
text, the charging functionality could be undertaken by software since it has relatively loose demands in terms of timing precision and does not interact with the CPU timer.

The third block that has to be investigated is the ERS. As it has already been stated, its role is to collect time-critical events associated with shaping and policing, and signal them to host. Charging, on the other hand, does not require a sophisticated response scheme. The following discussion is focused on shaping; however, the basic idea is valid for policing as well. The ERS can be designed either for increased timing accuracy or for scalability. In both cases it uses an event register of size \( N \) bits, where \( N \) is the number of input streams under traffic control. Bit \( i, 0 \leq i \leq N - 1 \), is set in this register as soon as the timer \( Tx_i \) counts down to zero (indication of an event for stream \( i \)). If the ERS is designed for accuracy, whenever at least one bit is set, an interrupt is issued to the host CPU. This asynchronous mechanism provides for timely reaction of the host CPU, but in the presence of many controlled streams it results in excessive interrupts.

We have identified three ways to handle this undesirable situation: (1) enhancement of the asynchronous method with an event threshold, (2) use of synchronous, i.e., clock-based, mechanism for events notification and (3) use of polling. The first approach suggests that there is an integer parameter identifying the number of timers \( Tx_i \) that need to expire before an interrupt is issued to the host. This parameter is introduced through the event_threshold register of Fig. 4 and can be modified in run-time. This enhancement allows the designer of the host CPU code to implement a module that monitors the rate of the interrupts and updates the value of the event_threshold register accordingly (this register is essentially used for interrupt throttling). By doing so, a balance between excessive interrupts and timing accuracy can be achieved. When increased accuracy is needed, the value of the register should be kept low, ideally 1. On the other hand, when processing load is the critical factor the value should be large. At any time instance the value of the event_threshold may range between 1 and \( N_o + 1 \), where \( N_o \) is the number of active streams. The largest value of this interval, i.e., \( N_o + 1 \), results in interrupt deactivation. This design enhancement is simple and powerful and, thus, it has been accommodated in the implementation of the ERS.

The second approach suggests that there exists a synchronous mechanism for interrupt activation. In this scheme, interrupts are allowed to reach the host CPU in periodic time intervals, e.g., every 200 \( \mu s \), provided that at least one event (i.e., \( Tx \) timer expiration) has occurred. The interrupt frequency can be adjusted in run-time by using a programmable timer. This technique has the advantages of the previous one (ease of implementation, control of processor load) and additionally, delivers a user-configurable upper bound on the time between the occurrence of an event and the interrupt activation.

The ERS design alternatives discussed so far use interrupts to signal the occurrence of an event to the host CPU. Although this method provides fast response, it does not exploit the advantages of the asynchronous communication in full, unless the host reaction (i.e., packet transmission or rejection) can be made interrupt-driven. This requires some modifications in the kernel code to handle these interrupts and trigger packet actions. If, on the other hand, the objective is to let the CPU scheduler take care of these actions, the use of polling gains significance. Our ERS design accommodates polling by setting the value of the event_threshold register to \( N_o + 1 \).

5. Development of a prototype board

In this section, we present the implementation of a peripheral board (hosted in a Linux PC) that supports traffic shaping, policing and charging by exploiting the building blocks of Fig. 4. The design of the peripheral board was done after partitioning the requirements to those that can be accommodated better in embedded software and to those that can be accommodated more efficiently in hardware. This hardware/software co-design requires not only a thorough study and dimensioning of the problem but also an estimate of the impact of the partitioning on the implementation cost (dominated by the cost of hardware), the delivered timing precision and the degree of programmability. The adopted approach is governed by the need to keep the cost of the board low without sacrificing the timing precision or the design’s potentiality to accommodate algorithmic enhancements.

5.1. System analysis and considerations

The traffic control operations of interest have been analyzed into a number of tasks listed in Table 1. Configurable hardware, microprocessors or a combination of them is a possible implementation of such a system. The decision whether to implement these tasks in hardware or software is usually based on major criteria, like silicon area consumption, required speed and frequency of use. However, in the current design, the final mapping onto hardware and software (presented in the last column of Table 1) was based on silicon area consumption alone, due to limitations in target hardware resources. Estimations about area consumption of tasks 3 and 5 were cross-checked with results derived after implementing part of the addressed functionality in hardware (see last paragraph of Section 5.5). The major tasks involve the management of communication mailboxes, the management of counters and timers, the execution of the traffic control algorithms and the implementation of (1).

The task of clock generation (activation of the free running clock of each timing block) and update is common to all traffic control functions (task 1 of Table 1) and is inherently accommodated by hardware. The communication between the host CPU and the peripheral board is done through requests and replies. All requests are stored in an input mailbox (dual port RAM) implemented in hardware.
Table 1
Classification of tasks according to implementation criteria and mapping for target H/W and S/W

<table>
<thead>
<tr>
<th>No.</th>
<th>Task</th>
<th>Silicon area consumption</th>
<th>Required speed</th>
<th>Frequency of occurrence</th>
<th>Mapping for target H/W-S/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clock generation/update</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>H/W</td>
</tr>
<tr>
<td>2</td>
<td>Storage of requests, management of input mailbox</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>H/W</td>
</tr>
<tr>
<td>3</td>
<td>Decoding of requests, calculation of (1)</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>S/W</td>
</tr>
<tr>
<td>4</td>
<td>Credit and charge accumulation</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>H/W</td>
</tr>
<tr>
<td>5</td>
<td>Monitoring of algorithms’ state, generation of functions’ outcome</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>S/W</td>
</tr>
<tr>
<td>6</td>
<td>Silence-enforcement and CPU notification</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>H/W</td>
</tr>
<tr>
<td>7</td>
<td>Storage of replies, management of output mailbox</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>H/W</td>
</tr>
</tbody>
</table>

(task 2) and are served in first-come-first-served basis. Proper dimensioning of the mailbox is necessary to ensure that no requests are lost due to overflow.

The decoding of the requests and the calculation of (1) is done by software (task 3). This choice facilitates the adoption of new traffic control algorithms by avoiding the large development and verification times that are inherent in hardware design and furthermore, as far as the modTB is concerned, it does not waste precious hardware resources for the implementation of the exponential function. The accumulation of both credit and charging data involves counting blocks and, thus, is undertaken by hardware (task 4). The monitoring of the state of the algorithms and the computation of the traffic control functions’ outcome are area consuming operations and, therefore, are mapped onto software (task 5). For traffic shaping, the enforcement of fine-grained silences prior to transmission and the notification of the CPU depend on the temporal resolution of the timing unit and thus, are undertaken by hardware (task 6).

The replies associated with the other two functions are transferred to the host CPU through an output mailbox (dual port RAM). However, the policing operation, being time critical as well, may alternatively use a combination of fast CPU notification and output mailbox. In this case, the ERS block of Fig. 4 is used to interrupt the host CPU, while the output mailbox is used to hold the outcome of the policing algorithm (i.e., permit or deny). The output mailbox, managed by hardware (task 7), can be also used to transfer debugging and statistics data to the host CPU.

5.2. System architecture

The architecture of the experimental card that has been designed to support the operations of traffic shaping, monitoring and charging is depicted in Fig. 5.

The card has been designed for the ISA computer bus and consists mainly of an FPGA (Xilinx 4085XLA), a microcontroller (Intel 80296SA) and memory modules. The choice of the ISA bus provides a good trade-off between speed and simplicity of the implementation, being completely consistent with the demands of an experimental design. The flash memory holds the microcontroller’s code. After initialization, the code is copied to static RAM for faster execution. Clock runs at 50 MHz and feeds the microcontroller. The clock-output port of the microcontroller feeds the FPGA with a clock running at 25 MHz; this clock acts as the master clock of Fig. 4. The card acts as a slave device on ISA bus and carries out zero-wait-state transfers of 16 bits. The lack of bulk transfers on the bus suggests that no DMA (direct memory access) transfers are needed. A brief description of the FPGA and the microcontroller (M296) blocks is given in the following.

5.3. The FPGA block

The internal block diagram of the FPGA is shown in Fig. 6. The ISA interface (i/f) block implements the ISA interface using the I/O (input/output) commands and decodes the ISA address bus in order to produce the chip-select signals for all internal blocks. The M296 i/f block implements the microcontroller interface and address decoding. There are two dual-port RAM blocks, namely the input mailbox and the output mailbox. The first (second) is write-only (read-only) on the side of the ISA and read-only (write-only) on the M296 side. N identical timing blocks (T_BLK) contain the appropriate counters, timers and registers for the realization of the timing unit. Each timing block contains a Clock Divider (CD) module which adjusts the clock frequency of the respective block by dividing the base clock (25 MHz) with a programmable value. The timestamp block consists of a counter fed by the FPGA’s master clock and is used for local timekeeping.

The “control + ERS” block implements the functionality of the ERS and generates interrupts on the ISA bus, as
soon as the number of the expired timers $T_x$ (reflected in the event register of the ERS) matches the value of the event threshold register (Section 4.2). In the current design, the event register is allowed to change value, i.e., to accommodate further timer expiration events, during the time between the activation of the interrupt line and the first access of this register by the host (the host CPU accesses this register during the interrupt acknowledge phase to identify the block that demonstrated the event). After the activation of the read signal, the event register freezes and further events are hold in a mirror register.

The design has been described in VHDL hardware description language and implemented in an XC4085XLA-09 Xilinx FPGA (Configurable Logic Blocks – CLB– matrix 56 × 56, Typical Gate Range 55,000–180,000, speed grade –09). The Leonardo Spectrum software package has been employed for synthesis and the Xilinx Foundation for place and route. The external or in-chip generated clock sources are limited to 8, due to the equal clock buffers residing inside the FPGA. Two of the clock buffers are used for external clock source buffering; this, ultimately, leaves us with $N = 6$ available buffers that are used to drive the different generated clocks.

The implementation results are depicted in Table 2. Each timing module requires about 6% of the chip’s hardware resources. Considering that external interfaces, control finite state machines and additional logic utilize 16% of the chip, one could theoretically put at most 14 timing modules within this specific FPGA.

All timing blocks are identical and can support any integer timing division of the master clock. Due to this repetition of one generic block, the design can be ideally mapped on to ASIC technology. Considering 11,000 gates to be a fairly equal gate count for the control finite state machines and interfaces, a large number of timing blocks can be implemented in a few square millimetres. In 0.18 micron technology, where 90,000 gates represent roughly one square millimetre, about 20 timing blocks would fit in.

5.4. The microcontroller block and the device driver

The software that has been developed to support the functionality of the card consists of the code written for the microcontroller and the code for the device driver. The major operations addressed by the microcontroller include the control of the timing unit and the mailboxes, the decoding of requests and the execution of the LB, TB and modTB algorithms for traffic shaping and policing. Supplementary functionality includes the construction of the look-up table and code for statistics collection and debugging.

The control of the timing unit includes provision for updating, at run-time, the clock divider and the event threshold registers of Fig. 4. The update of the clock prescaler provides for dynamic adjustment of the per-stream allocated (effective) rate. This operation is triggered by the host and can be based either on a local decision or on feedback from other network elements. In the current design, the adjustment of the frequency of the generated interrupts is initiated by the host and carried out by the microcontroller. However, if we allow this adjustment to be based on a decision made by the microcontroller, it can be done in a way that is transparent to host.

In shaping and policing with the modTB, the calculation of the expression inside the round brackets in (1) is accelerated by a look-up table eliminating the complexity associated with the exponential function and the division operation. Furthermore, the software includes code for statistics (e.g., number of packets subjected to traffic control) and debugging. The latter is accomplished by using either

Table 2

<table>
<thead>
<tr>
<th>Xilinx XC4085XLA-09 FPGA implementation results</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Overall utilization</td>
</tr>
<tr>
<td>Overall CLBs (Gate equivalent)</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
</tr>
<tr>
<td>No. of T_BLKs</td>
</tr>
<tr>
<td>CLBs/T_BLK</td>
</tr>
<tr>
<td>% Utilization of T_BLKs (CLBs)</td>
</tr>
<tr>
<td>% Utilization of finite state machine and other logic (CLBs)</td>
</tr>
</tbody>
</table>
the input and output mailboxes or the microcontroller’s serial port.

The device driver has been written for the Linux operating system and is dynamically linked into the kernel. It accesses memory locations allocated to the peripheral card and handles interrupts. The card reserves a portion of the x86 I/O mapped address space (addresses 0x300 to 0x30b) and an interrupt line (IRQ) of the programmable interrupt controller. The 12 reserved byte-addresses are allocated to the event register of the ERS (two addresses, corresponding to a 16-bit wide register), the input and output mailboxes (two addresses each), the register that holds the size of the valid data of the output mailbox (two addresses) and the timestamp counter (four addresses).

5.5. System overheads

The overhead associated with the use of the card is a combination of software execution time, hardware delays and the time a request must wait before receiving service by the microcontroller’s CPU (queueing time). The sequence of events during shaping and the associated delays and overheads are depicted in Fig. 7. As far as the other two functions are concerned, the charging function involves only bus, queuing, processing and polling (on the side of host CPU) overhead (since the replies are communicated to the host through the output mailbox), and the policing operation has no enforced-silence delay.

Supposing that a request is sent from the host at time instance $t_1$ (first access of the I/O mapped input mailbox), the extra overhead inserted by the card is $(t_{21} - t_{54})$, where $t_{ij} = t_i - t_j$. The overhead $t_{21}$ depends on the speed of the interconnection bus and the size of the request (two 16-bit words for the shaping operation). The overhead $t_{32}$ corresponds to queuing time and involves the time required for the processing of the precedent requests and the time associated with the polling of the input mailbox. For a given processor, the overhead $t_{43}$ depends on the computational complexity of the traffic control algorithm. For example, the implementation of the modTB, using a look-up table to accelerate the calculation of (1), reported an overhead of approximately 150 μs which is due, to a large extent, to the two multiplications involved in that formula. On the other hand, the overhead associated with the charging operation was substantially less, since no mathematical calculations are involved in this function. The overhead $t_{45}$ corresponds to the host processor interrupt latency (about 15 μs in standard Linux for a moderately loaded system). Finally, the overhead $t_{76}$ corresponds to the execution of the Interrupt Service Routine (ISR) in the host processor which starts with the acknowledgment of the IRQ interrupt signal.

The above analysis indicates that the extra overhead mainly manifests itself as overhead $t_{42}$. However, experiencing with the board in real conditions showed that upon arrival of a new request the input mailbox was empty with very large probability and the microcontroller was engaged in the polling loop. This suggests that the overhead $t_{42}$ can be safely ignored. The overhead $t_{43}$ depends on the executed operation (e.g., the code for traffic shaping is lengthier than the code for charging) and the frequency, with which interrupt requests are issued to the microcontroller. These requests reach the microcontroller through the IRQm signal in Fig. 7 and are used for control purposes. When increased accuracy is needed, the overhead $t_{43}$ can be compensated by counting it in the value of the enforced silence.

The extra overheads introduced by the processing time and the interface between the timing and the AEC units can be minimized if the software functionality is integrated into hardware. This step would improve the performance of the traffic control module and facilitate the interface with existing networking hardware. Indeed, we made an initial attempt to move a part of the microcontroller’s functionality into hardware and investigate the achieved gain. This functionality consisted of an elementary request-decoding unit, a look-up table for a fixed QoS parameter (i.e., the input to the look-up table is the packet size and the output is the silence given by (1)) and the non-linear data spacing part of the modTB. The hardware latency was measured by recording the time a shaping request for a packet of zero size was sent to the card and the time the response (activation of the interrupt line) was received. The result showed that the time overhead for shaping was reduced to less than 5 μs. This attempt illustrates, to a large extent, the dynamics of a unified architecture and deserves further investigation.

6. Experimental results

The target of the experiments is twofold: first, to verify the capability of the proposed architecture to enforce fine-grained time silences between transmissions of packets and second, to demonstrate these benefits in the context of the modTB shaping scheme that provides statistical QoS guarantees. The reason we evaluate the modTB is because its implementation is more complicated than that of simple leaky buckets and token buckets and, thus, it can derive...
much benefit from a hardware timing unit and an integrated hardware/software design.

The experiments took place on a laboratory network and involved real IP traffic transmitted by hosts attached to 100 Mbps Ethernet segments (every host was equipped with an experimental card). The (end) hosts were set to generate traffic, which was, in turn, shaped by the peripheral boards, prior to transmission in the network. However, this does not mean that the traffic control module is intended (just) for end systems. If the unshaped traffic (i.e., the traffic offered to the board) is interpreted as (emulated) incoming network traffic, the end hosts can be logically turned into routers (e.g., edge routers in a DiffServ domain).

In order to avoid unpredicted delays originating from traffic generation at user level, we developed a software code capable of generating IP packets in an efficient manner. This traffic generator consists of two parts: a user-level program and a kernel module. For each controlled stream, the user level software generates samples of packet sizes according to statistical distributions and transfers them to kernel space in a batch mode. The kernel module takes as input the packet size, sends a shaping request to the card, constructs an IP packet of that packet size and dummy payload, appends the link layer header and stores the frame in memory. Upon reception of the response, indicating that the appropriate silence has been enforced, the kernel module delivers the frame to the network device driver and continues with the next packet. After all samples are consumed by the kernel module, the user program is instructed to provide a new set. In both experiments, the responses are indicated to host through interrupts and the ERS is configured for maximum timing precision.

The development of this specialized traffic generation software provides us with better control over the traffic manipulation process (traffic classification into streams, storage prior to transmission, frame delivery to network device driver, etc) and eliminates the need to interfere with the networking layers of the operating system (this would be necessary if we considered traffic stemming from real applications).

6.1. Temporal resolution and CPU load

The first experiment assumes that an IP traffic stream is subjected to shaping according to the modTB algorithm. A further assumption is that all packets in the stream have constant size equal to 240 bytes (a constant size is preferable in this experiment because it gives better illustrative results) and that the shaper’s queue is never emptied (the shaper never stays idle, i.e., after the enforcement of the silence and prior to the transmission of the packet a new shaping request is sent to the card). The elementary data unit (DA) is set to 16 bytes; the packets size is then 6 = 15DA. Furthermore, the silence that is enforced prior to (or after) the transmission of a packet is always given by (1), i.e., minimum and maximum credit thresholds are equal to zero. This configuration corresponds to maximum utilization scenario and tight shaping. The QoS parameter, q, is set according to the requirement Pr(queue_length > 1000DA) = 10−3 (see Section 2.2). The clock prescaler of the 25 MHz master clock is set to 313, so that the enforced silence is 200μs (the value was chosen so as to facilitate direct comparisons with the results of Section 3). This silence is given by (1); however, the quantity inside the square brackets is now rounded to the nearest integer.

The inter-packet distances were measured using the tcpdump tool and the PDF of the difference between the actual and desired distances is depicted in Fig. 8 (the tcpdump software works with microsecond resolution which is high enough to make strong statements about the fine-time-scale properties of the traffic stream). The processing overhead, (t = t) with respect to Fig. 7, has been counted in the enforced silence; otherwise, processing overhead and computed silence would act cumulatively. The parameters of the experiment correspond to 5000 interrupts per sec and the data rate, including the link layer header, is approximately 10.2 Mbps. The results confirm that the accuracy of the timing unit is high, given that 95.8% of the values lie in the interval 200 ± 5μs and 99.6% of them in the interval 200 ± 10μs (the reader is referred to Fig. 3 for a comparison with implementations using software-based timers). The fluctuation of the inter-packet distances is mainly due to the variability of the interrupt response time of the host CPU (and, thus, it cannot be avoided). We would come up with a similar figure if we recorded the time intervals between the interrupts issued by the card. However, the strength of the result of Fig. 8 is that the accuracy is well preserved even at the time the packets are transmitted in the network.

The previous experiment, which evaluated the temporal resolution of the timing unit, involved only one active stream. It is expected however, that the presence of a large number of active streams will have a heavier impact on the difference between scheduled and desired times than that presented in Fig. 8. The main reason for this is that interrupts issued to host CPU may block for various periods of time waiting for the completion of the interrupt service routine that is already in process. Such side effects are

![Fig. 8. Distribution of the difference between actual and desired inter-packet distance (μs)](image-url)
common in interrupt-driven implementations and can be hardly avoided.

The contribution of the experiment to the host CPU load involves only system (kernel) load and can be divided in two parts: (a) the load associated with the use of the card for traffic shaping and (b) the load associated with the frame construction and transmission. The experiment does not contribute to user load since the packets involved in shaping are of constant size and thus, there is no interaction between the user-level program (traffic generator process) and the kernel module. The system overhead was measured on a 133 MHz Intel Pentium I machine running standard Linux 2.2.18. The results were obtained by the top command (averaging over many outcomes) and showed that the experiment contributed to the system load by less than 1%. It is anticipated that the dominant component of the CPU load associated with the use of the peripheral board is context switching due to interrupts, while the other component, namely the access of the computer peripheral bus, contributes much less to the CPU usage. However, a more sophisticated measurement tool is required in order to evaluate this hypothesis and measure the exact amount of the system overhead imposed by the off-board implementation of the traffic control functionality.

### 6.2. Exploiting the timing unit for traffic control: shaping with QoS guarantees

The capability of the hardware/software co-design to support data spacing with stringent timing requirements was demonstrated through experimentation with a single stream. In this experiment, we loose the restriction for fixed-length packets and constant inter-packet spacing and use the peripheral card to shape a larger number of separate streams according to the modTB algorithm. The experiment evaluates the ability of this scheme to provide statistical QoS guarantees (for experimental results using software-based timers and on-board implementation of the traffic shaping functionality, see [29,30]).

The experimental setup involves three standard end systems (PCs) acting as sources of shaped IP traffic. For this purpose, each one of them is equipped with an experimental board that can support up to six independently shaped streams. All shaped streams are transmitted through a 100 Mbps Ethernet switch to a Cisco 7200VXR router running the IOS 12.0 operating system. However, due to inability to sample the buffers of the router interfaces (IOS 12.0 features an incomplete management information base) we rely on a simple workaround, according to which the traffic is relayed from the router’s fast Ethernet interface to its 155 Mbps ATM interface and is forwarded to a FORE ASX-200 ATM switch. The IP over ATM segmentation and encapsulation mechanism undertaken by the router inevitably increases the size of the transmitted information. However, this overhead is negligible and can be safely ignored. The output port of the ATM switch sup-

<table>
<thead>
<tr>
<th>Source</th>
<th>Parameters</th>
<th>Packet length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td>Mean = 65</td>
<td>Min = 5, Max = 255</td>
</tr>
<tr>
<td>Uniform</td>
<td>Min = 5, Max = 255</td>
<td></td>
</tr>
<tr>
<td>Pareto</td>
<td>x = 1.2, k = 20</td>
<td>Min = 20, Max = 255</td>
</tr>
</tbody>
</table>

ports rate limiting and sampling of its queue at the sufficient rate of 100 Hz.

Three types of traffic sources were used in the experiment and their parameters are presented in Table 3. (The Pareto distribution is defined by: \( PDF(x) = 0, x \leq k \), and \( PDF(x) = 1 - (k/x)^a, x > k; a > 0. \) The last type of traffic is described by a heavy-tailed distribution resulting in long-range dependence and (asymptotic) self-similarity [31]. The elementary data unit, DA, is set as before to 16 bytes and the maximum size of the IP packet to 255DAs, thus requiring 8 bits for its representation (IP packets of arbitrary maximum length can be supported through appropriate adjustments of the DA value). The initial and maximum credit thresholds have been set to 255 and 300 DAs, respectively, to emulate real conditions and to allow a small number of packets to be transmitted with linear silence. The experiment assumes maximum utilization, i.e., the shapers have always data to transmit and thus, the credit counters never saturate. Each of the shaped streams is allocated an effective rate \( f_i = 1.66 \text{ Mbps} \) (the clock prescaler is 1928) and the QoS parameter, \( q \), is set as in Section 5.1 (\( q = -0.006908 \text{ DA}^{-1} \)).

The distribution of the queue-length of the ATM switch output port for 12 and 18 multiplexed shaped streams is depicted in Fig. 9. In any case, the service rate of the switch output port is set to the sum of the effective rates of the input streams, i.e., \( C = 20 \text{ Mbps} \) for 12 sources and \( C = 30 \text{ Mbps} \) for 18 sources. The slopes of the experimental curves approximate well the slope of the theoretical curve obtained from the QoS requirement. Visible deviations take place in large queue lengths and are mainly due to the small number of multiplexed streams.

### 7. Conclusions and future work

The performance of many traffic control operations depends on the efficiency of the timing unit found in the sys-
tem that implements them. The design of a timing unit strives to achieve high precision (temporal resolution), good scalability with a large number of controlled streams and manageable implementation complexity. This paper addressed the design, implementation and evaluation of a programmable timing unit that is able to cope with the stringent timing demands of several traffic control functions implemented in the nodes of experimental packet-switched networks. The validation of the timing unit, which was implemented in hardware, was done through the development of a prototype board that accommodated the algorithmic parts of the traffic control functions in embedded software. The experiments demonstrated that the timing unit provides a temporal resolution of a few microseconds without imposing high overhead on host CPU and that a well-balanced software/hardware co-design is able to deliver efficient implementations for demanding traffic control algorithms.

Future work will continue to investigate ways of moving parts of the traffic control functionality, currently undertaken by software, to hardware. Towards this direction, a great challenge for the designer will be the control of the silicon area consumption, especially as far as implementations in configurable hardware are concerned. The integration of the functionality will deliver a single chip, which, in turn, can be used in the design of network adapters (possibly accommodating several physical interfaces) that will include support for distributed traffic control. It is our belief that such implementations are of great value in experimental platforms.

References

embedded systems, traffic characterization and analysis and QoS support for IP networks. He is member of the IEEE and of the Technical Chamber of Greece.

Nikos Moshopoulos received his diploma in Computer Engineering from the University of Patras and his Ph.D. from the Electrical and Computer Engineering department of the National Technical University of Athens. Since 2001 he is with the ATME, a world leader in advanced semiconductors industry. He has been managing a number of ASIC R&D projects in the area of telecommunications. He is the author or co-author of a vast amount of papers in journals and conferences. His research interests include cryptography in hardware, advanced integrated circuit design, and low-power techniques.

Dimitrios Loukatos received the Diploma in Electrical and Computer Engineering and the Ph.D. degree in Telecommunications and Computing, both from the National Technical University of Athens (NTUA), in 1997 and 2002, respectively. He is currently research associate with the Electrical Engineering School of NTUA. His research interests include computer systems design, communication traffic modeling and analysis, mobile, and wireless networking. He is member of the Technical Chamber of Greece since 1998.

Kostas Marinis received a BEng (Honors) in Computer Systems Engineering and an M.Sc. in Digital Electronics and Parallel Processing Systems from the University of Sussex, UK, in 1997 and 1998, respectively. From 1999 to 2000 he worked at Madge Networks, UK, as a Hardware Development Engineer. In 2000, he joined the Microprocessors and Digital Systems Lab, National Technical University of Athens, as a research assistant, where he worked on efficient hardware implementations of cryptographic algorithms. From 2002 to 2005, he was also with Atmel Hellas S.A., working as IC Design engineer. He joined ESTEC/ESA, the Netherlands, as a contractor on September 2005, where he currently works as a Microelectronics Engineer. His research interests include Systems-on-Chip, IP cores design, and reuse methodologies, processor architectures, and on-chip interconnect architectures.

Panagiotis Stathopoulos received his Electrical and Computer Engineering Degree (1999) and his Ph.D. (2004) from the Electrical and Computer Engineering School of the National Technical University of Athens (NTUA), Greece. His Ph.D. thesis was in the area of next generation heterogeneous wireless IP networks. From 1999, he has participated in a number of European Union and nationally funded large scale research projects. His main research interests are applications, services and architectures for next generation networks. He has been an adjunct Lecturer at the University of the Aegean. He is currently a senior researcher within the Computer Networks Laboratory of NTUA while he teaches at the Athens Technological Education Institute.

Nikolas Mitrou received the Diploma degree in Electrical Engineering from the National Technical University of Athens (NTUA) in 1980, the M.Sc. degree in Systems and Control from the UMIST, Manchester, in 1982, and the Ph.D. degree in Electrical Engineering from NTUA in 1986. He is now a full professor in the School of Electrical and Computer Engineering of NTUA. His research interests are in the areas of digital communications, communication networks and networked multimedia in all range of studies: design, implementation, modeling, performance evaluation and optimization, applications. Prof. Mitrou is member of the IEEE and member of the IFIP (TC6).